案例 14-基于边界表示的区域扫描线填充算法

文档编写：霍波魏

校稿/修订：孔令德

时间 2019~2020

联系方式：QQ997796978

说明：本套案例由孔令德开发，原版本为 Visual C++6.0，配套于孔令德的著作《计算机图形学-基于 MFC 三维图形开发》一书。孔令德计算机工程研究所的学生霍波魏在学习计算机图形学期间，对本套案例进行了升级并编写了学习文档。现在程序的编写和程序的解释都是基于 Windows 10 操作系统，使用 Microsoft visual studio 2017 平台的 MFC（英文版）开发。

一、知识点

扫描线种子填充算法的原理是先将种子像素入栈，种子像素为栈底像素，然后判断栈是否为空，如果不为空，执行下列操作：

(1)栈顶像素出栈

(2)沿扫描线对出栈像素的左右像素进行填充，直到遇到边界像素为止。即每出栈一个像素，就对区域包含该像素的整个连续区间进行填充。

(3)同时记录该区域，将区域最左端像素记为 X_{left},最右端像素记为 X_{right}。

(4)在区间[X_{left}, X_{right}]中检查与当前扫描线相邻的上下两条扫描线的有关像素是否全为边界像素或已填充像素，若有非边界且未填充的像素，则把未填充区间的最右端像素取作种子像素入栈。

二、案例描述

本案例用边界表示的扫描线填充算法，填充“书香”的空心汉字区域。

三、实现步骤

1. 添加栈结点 CStackNode 类。
2. 在 CTestView 类中定义出/入栈函数、填充函数以及双缓冲绘图函数。
3. 在 CTestView 类的 ScanLineFill 函数中。
4. 在 DoubleBuffer 函数中绘图并在 OnDraw 函数中进行调用。

四、主要算法

CTestView 类

```
public:
  void DoubleBuffer(CDC* pDC);//双缓冲绘图
  void ScanLineFill();//区域扫描线填充算法
  void Push(CPoint point);//入栈
  CPoint Pop();//出栈
protected:
  int nClientWidth, nClientHeight;//屏幕客户区宽度和高度
  int nHWidth, nHHeight;//屏幕客户区的半宽和半高
  CStackNode *pHead, *pTop;//结点指针
  CPoint Seed;//种子
  COLORREF SeedClr, BoundaryClr;//种子色和边界色
void CTestView::ScanLineFill()//区域扫描线填充算法
{
  CDC* pDC = GetDC();
  BoundaryClr = RGB(0, 0, 0);//边界颜色
  bool bSpanFill;
  pHead = new CStackNode;//建立栈结点
  pHead->pNext = NULL;//栈头结点指针域为空
  Push(Seed);//种子像素入栈
  //射线探测种子位置法
  CPoint p;//p用于判断种子与图形的位置关系
  p.x = Seed.x - 1;
  while (BoundaryClr != pDC->GetPixel(p.x, Seed.y) &&
          SeedClr != pDC->GetPixel(p.x, Seed.y))//左方判断
  {
      p.x--;
      if (p.x <= 0)//到达客户区最左端
      {
          MessageBox(_T("种子不在图形之内"), _T("警告"));
          return;
      }
  }
  p.y = Seed.y - 1;
  while (BoundaryClr != pDC->GetPixel(Seed.x, p.y) && SeedClr !=
pDC->GetPixel(Seed.x, p.y))//上方判断
  {
      p.y--;
      if (p.y <= 0)//到达客户区最上端
      {
          MessageBox(_T("种子不在图形之内"), _T("警告"));
              return;
      }
  }
  p.x = Seed.x + 1;
```

```
while (BoundaryClr != pDC->GetPixel(p.x, Seed.y) &&
         SeedClr != pDC->GetPixel(p.x, Seed.y))//右方判断
{
    p.x++;
    if (p.x >= nClientWidth)//到达客户区最右端
    {
        MessageBox(_T("种子不在图形之内"), _T("警告"));
        return;
    }
}
p.y = Seed.y + 1;
while (BoundaryClr != pDC->GetPixel(Seed.x, p.y) &&
         SeedClr != pDC->GetPixel(Seed.x, p.y))//下方判断
{
    p.y++;
    if (p.y >= nClientHeight)//到达客户区最下端
    {
        MessageBox(_T("种子不在图形之内"), _T("警告"));
        return;
    }
}
int xLeft, xRight;//区间最左端与最右端的像素
CPoint PopPoint, PointTemp;//出栈像素和临时像素
while (pHead->pNext != NULL)//如果栈不为空
{
    PopPoint = Pop();
    if (SeedClr == pDC->GetPixel(PopPoint.x, PopPoint.y))
        continue;
    PointTemp = PopPoint;
    //处理当前扫描线
    while (BoundaryClr != pDC->GetPixel(PointTemp) &&
        SeedClr != pDC->GetPixel(PointTemp))
    {
        pDC->SetPixelV(PointTemp, SeedClr);
        PointTemp.x++;
    }
    xRight = PointTemp.x - 1;//获得扫描线上最右端的像素
    PointTemp.x = PopPoint.x-1;
    while (BoundaryClr != pDC->GetPixel(PointTemp) &&
            SeedClr != pDC->GetPixel(PointTemp))
    {
        pDC->SetPixelV(PointTemp, SeedClr);
        PointTemp.x--;
    }
```

```
        xLeft = PointTemp.x + 1;//获得扫描线上最左端的像素
        //处理下一条扫描线
        PointTemp.x = xLeft;
        PointTemp.y += 1;
        while (PointTemp.x < xRight)
        {
            bSpanFill = FALSE;
            while (BoundaryClr != pDC->GetPixel(PointTemp) &&
                     SeedClr != pDC->GetPixel(PointTemp))
            {
                bSpanFill = TRUE;
                PointTemp.x++;
            }
            if (bSpanFill)
            {
                PopPoint.x = PointTemp.x - 1;
                PopPoint.y = PointTemp.y;
                Push(PopPoint);
                bSpanFill = FALSE;
            }
            while ((BoundaryClr == pDC->GetPixel(PointTemp) && PointTemp.x <
xRight) || (SeedClr == pDC->GetPixel(PointTemp) && PointTemp.x < xRight))
                PointTemp.x++;
        }
        //处理上一条扫描线
        PointTemp.x = xLeft;
        PointTemp.y -= 2;
        while (PointTemp.x < xRight)
        {
            bSpanFill = FALSE;
            while (BoundaryClr != pDC->GetPixel(PointTemp) &&
                SeedClr != pDC->GetPixel(PointTemp))
            {
                bSpanFill = TRUE;
                PointTemp.x++;
            }
            if (bSpanFill)
            {
                PopPoint.x = PointTemp.x - 1;
                PopPoint.y = PointTemp.y;
                Push(PopPoint);
                bSpanFill = FALSE;
            }
            while ((BoundaryClr == pDC->GetPixel(PointTemp) && PointTemp.x <
```

```
xRight) || (SeedClr == pDC->GetPixel(PointTemp) && PointTemp.x < xRight))
                PointTemp.x++;
        }
    }
    delete pHead;
    pHead = NULL;
    ReleaseDC(pDC);//释放DC
}
```

五、实现效果

区域边界表示的扫描线填充算法效果如图 14-1 所示。

图 14-1 区域边界表示的扫描线填充算法效果图

六、案例心得

该案例采用的是扫描线种子填充算法，因为单纯的使用种子填充算法会降低算法的效率，同时会占用大量的存储空间，所以当填充区域较大时，可以使用扫描线种子填充算法，可以节省存储空间，提高填充效率。